Java Programming 2 – Lab Sheet 7

This Lab Sheet contains material based on Lectures 13—14.

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 14 November 2019.

Aims and objectives

- Overriding **equals()** and **hashCode()** in a class
- Writing event handlers using Swing

Set up

1. Download **Laboratory7.zip** from Moodle. **(This file will be provided on Friday.)**
2. Launch Eclipse as in previous labs (see the Laboratory 3 lab sheet for details)
3. In Eclipse, select **File** → **Import** … (Shortcut: **Alt-F, I**) to launch the import wizard, then choose **General** → **Existing Projects into Workspace** from the wizard and click **Next** (Shortcut: **Alt-N**).
4. Choose **Select archive file** (Shortcut: **Alt-A**), click **Browse** (Shortcut: **Alt-R**), go to the location where you downloaded `Laboratory7.zip`, and select that file to open.
5. You should now have one project listed in the Projects window: **Lab6**. Ensure that the checkboxes beside this project is selected (e.g., by pressing **Select All** (Shortcut: **Alt-S**) if it is not), and then press **Finish** (Shortcut: **Alt-F**).

Submission material

Again, this lab builds on the work we have done in previous labs. As part of the starter code, you have been provided with the updated **Monster** and **Trainer** classes from Lab 6, as well as GUI classes to display and manipulate **Trainer** objects.

Your tasks are as follows:

- To update the **Monster** class so that it overrides **equals()** and **hashCode()** properly
- To write the callback methods in the **TrainerFrame** and **AddMonsterDialog** classes to implement appropriate behaviour.

Overriding equals() and hashCode()

You should implement appropriate **equals()** and **hashCode()** methods in **Trainer**, **Monster**, and **Attack**. For all classes, equality should be defined based on all of the fields. It is fine to use the auto-generated Eclipse functions to add these methods.

The provided JUnit tests will verify that your implementations behave as required.

Implementing GUI behaviour

The code you have been given in the **gui** package is the start of a Swing "trainer manager" application. If you run the main method of the **TrainerFrame** class, you will see a window pop up similar to the following:

Note that when you click on the buttons, nothing happens at the moment. Your job is to write the back-end code to turn this window into an interactive GUI application.

Internal structure of TrainerFrame class

There is a lot of code in the **TrainerFrame** class, much of it involved in actually laying out the user interface. Here is a summary of the overall structure:

- Each graphical component on the screen is an instance field – for example, the **tradeButton** field represents the "Trade" button
- The two lists of monsters are represented by the **JLists** called **trainer1List** and **trainer2List**
- The underlying lists of displayed monsters are stored in the **trainer1List** and **trainer2List** fields, which are of type **DefaultListModel<Monster>**.
    - Note: as mentioned in the lecture **DefaultListModel** is very similar to **ArrayList** – it provides methods for adding and removing elements and other similar operations. The method names are a bit different from **ArrayList**, though, so you might need to read the documentation at https://docs.oracle.com/javase/8/docs/api/javax/swing/DefaultListModel.html for details
- The **TrainerFrame** constructor creates all of the components and lays them out on the screen, and also sets up the behaviour of the top-level window – e.g., it ensures that the program exits when the window is closed.
- **TrainerFrame** also implements **ActionListener**, which means that it provides an **actionPerformed()** method. This method is registered with all of the on-screen buttons, which ensures that whenever any of the buttons is pressed, the **actionPerformed()** implementation is called.

Behaviour to implement

The following is the required behaviour for each button; I will give more suggestions on how to implement each behaviour below:

- **Add**: when one of these buttons is clicked, you should create a new instance of the provided **AddMonsterDialog** class and call its **setVisible()** method; the required behaviour of this dialog will be specified below.

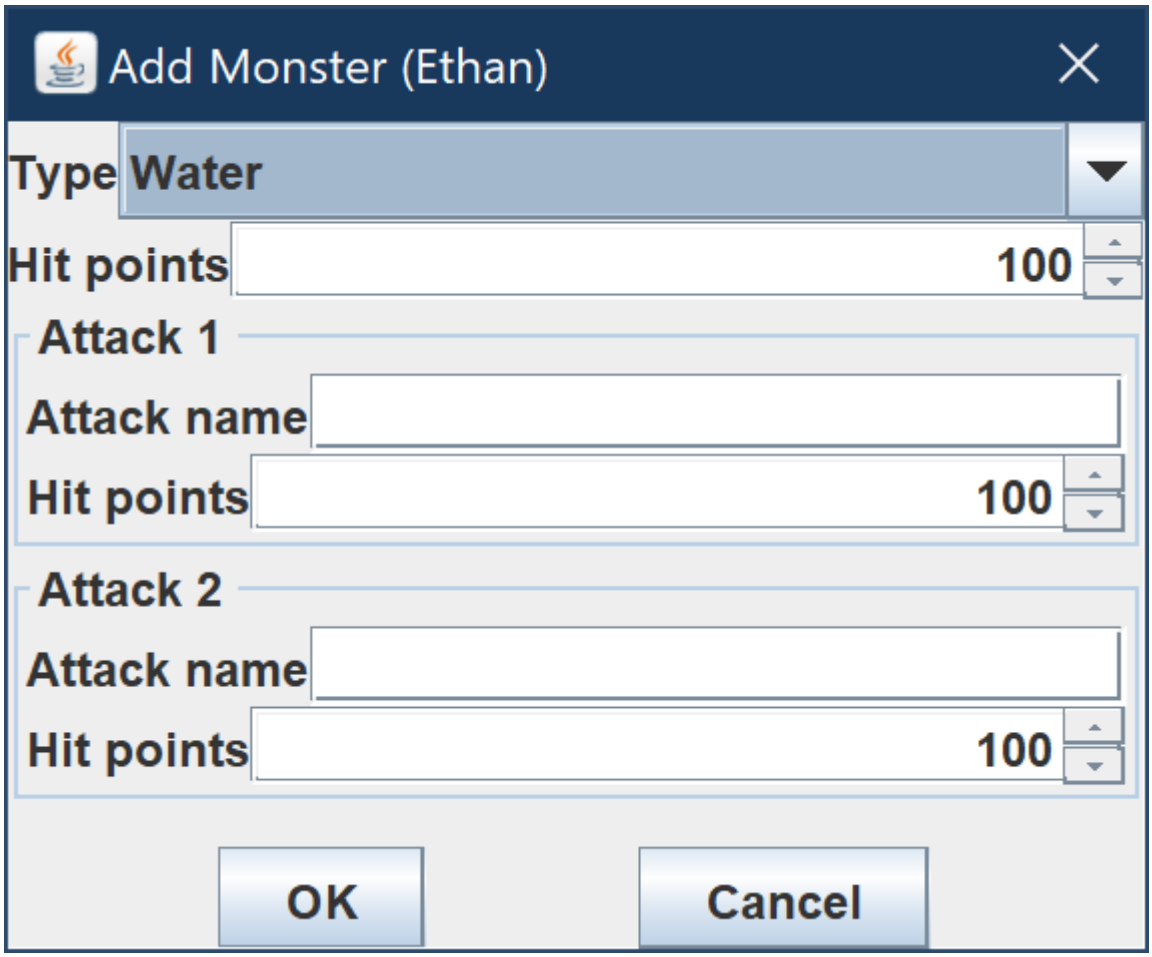
- **Delete**: when one of these buttons is clicked, you should check whether a monster is selected in the relevant list. If none is selected, the button should do nothing; if a monster is selected, then it should be removed from the displayed list and the Trainer's list.
- **Trade**: when this button is clicked, you should check that a monster is selected in both trainer's lists. If two monsters are selected, you should create a new **Trade** object and execute the trade; you should also update the two displayed lists to show the new list of monsters for both trainers.

All of the above behaviour should be implemented inside the **TrainerFrame.actionPerformed()** method. Inside that method, you can use **event.getSource()** to find out which of the buttons was actually pressed so that you can implement the required behaviour.

Note that I have already provided an implementation for the "delete" behaviour; you must add the behaviour for the other buttons.

Adding a Monster

As mentioned above, the process of adding a Monster to the list should be handled in the **AddMonsterDialog** – when one of the "Add" buttons is pressed, the **MonsterFrame** should just create a new **AddMonsterDialog** with appropriate parameters and make it visible through **setVisible(true)**. Here is what it looks like on screen:



The following is the structure of **AddMonsterDialog**:

- The associated **Trainer** and **DefaultListModel** fields are stored in the fields **trainer** and **model**, which allows the **AddMonsterDialog** to modify the list of Monsters when needed (see below).
- Each graphical component is represented by a field – e.g., **attack1Field** is the text field for the name of the first attack.
- The "Type" combo box specifies the monster type, which can be Electric, Water, or Fire.
- **AddMonsterDialog** implements **ActionListener**, which means that it also has an **actionPerformed** method. This method is called whenever the "OK" and "Cancel" buttons are pressed.

A skeleton **actionPerformed** method has been created for you, and I have already implemented the behaviour for the "Cancel" button (the dialog box is closed with by calling **dispose()**).

You should implement the following behaviour to respond to a press on the "OK" button: a new **Monster** of the appropriate type should be created based on the data entered into the fields (you may want to use the static **Trainer.createMonster()** in the provided starter code). The newly created **Monster** should then be added to the given **Trainer** object, and also to the **DefaultListModel** so that it is displayed properly, and the dialog should be closed afterwards using **dispose()**.

Testing your code

As in the previous labs, a set of JUnit test cases are provided to check the behaviour of your classes, in the file **test/TestLab7.java** – please see the lab sheet for Lab 4 for instructions on using the test cases. You can use the test cases to verify that your code behaves as expected before submitting it. But note in particular that **the test cases only test your equals() and hashCode() behaviour** – you will need to test your GUI code yourself manually.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not. Before submission, make sure that your code is properly formatted (e.g., by using **Ctrl-Shift-F** to clean up the formatting), and also double check that your use of variable names, comments, etc is appropriate. **Do not forget to remove any "Put your code here" or "TODO" comments!**

When you are ready to submit, go to the JP2 moodle site. Click on **Laboratory 7 Submission**. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – probably **M:\eclipse-workspace\Submission7\src** -- and drag only the *five* Java files **monster/Monster.java, monster/Attack.java, trainer/Trainer.java, gui/TrainerFrame.java** and **gui/AddMonsterDialog.java** into the drag-and-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the .java file is uploaded to the system. Then click **submit assignment** and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range “Excellent” (*****) to “Very poor” (*). Example scores might be:

5*: you completed the lab correctly with no bugs, and with correct coding style

4*: you completed the lab correctly, but with stylistic issues – or there are minor bugs in your submission, but no style problems

3*: there are more major bugs and/or major style problems, but you have made a good attempt at the lab

2*: you have made some attempt

1*: minimal effort

Possible (optional) extensions

Here are some additional things to try if you want further practice with GUI programming:

- At the moment, all of the buttons are always enabled, whether the associated action can actually be executed or not. See if you can work out how to enable/disable the buttons dynamically depending on what the user has selected (e.g., **Trade** should only be active when one monster is selected on each side)
- The ListModels and the Trainer objects are managed separately in the code at the moment, meaning that all actions need to be done twice, once to change the Trainer and once to change the display. Update the code to link the ListModels directly to the Trainers (hint: you probably want to subclass **AbstractListModel**)